Système d'exploitation II

TP 2 : Héritage et communication entre Processus

Le présent TP a pour objectif l'étude des attributs des processus et des mécanismes de communication entre les processus hiérarchiques.

<u>Partie I : Attributs des Processus :</u>

Dans cette partie nous traiterons les attributs des processus sous linux. Pour cela nous utiliserons les primitifs systèmes de récupérations des attributs :

Getpid ()			identité du processus en cours
Getppid ()			identité du processus parent de celui en cours
Getuid ()			propriétaire réel : en général celui du login
geteuid ()		propriétaire effectif : propriétaire du processus pour lequel le bit set-uid a été modifié pour permettre à un exécutable d'être exécuter par un autre utilisateur.
set-uid est le bit à changer pour changer d'utilisateur en utilisant la fonction setuid
getgid ()		groupe réel
getegid ()		groupe propriétaire effectif
getpwd ()		répertoire de travail

- Sous le dossier home associé à la session courante, créez un répertoire intitulé TP2.
- Ecrire un programme C « prog1.c » sous TP2, qui permet de créer un processus fils.
- Modifier « prog1.c » afin d'afficher les attributs de chaque processus (père et fils), le **pid**, **ppid**, le **uid, guid** et le répertoire de travail. Pour la récupération du répertoire en utilise la syntaxe suivante : printf(" répertoire de travail : %s\n ",getcwd(buf,1024)), avec buf est une variable de type chaine de caractère.
- Quelles sont les attributs différents entre les deux processus. Expliquer.

<u>Partie II : Passage des variables :</u>

Dans cet exercice nous proposons d'étudier le passage de variable entre le processus père et fils.

- Implémenter un programme (prog2.c) qui permet de créer un processus fils et père.
- Dans le même programme, déclarer une variable **m** de type entier, avant d'accéder aux zones de code des deux processus. afficher l'adresse et le contenu de la variable dans les deux processus. Qu'est ce que vous remarquez ?
- Exécuter le programme suivant (prog3.c) :

```
#include <sys/times.h>
int n=1000;
        main(){int m=1000, pid;
        printf("Adresse de n dans le père: %p\n ", &n);
        printf("Adresse de m dans le père: %p\n ", &m);
        printf("1a valeur de m et n dans le père : %d %d\n ", m, n);
```

```
switch(pid=fork()){
    case        -1:     perror("fork");exit(2);
    case        0 :                         /* on est dans le processus fils*/
                        printf("Adresse de n dans le fils: %p\n ", &n);
                        printf("Adresse de m dans le fils: %p\n ", &m);
                        printf("2 valeur de m et n dans le fils : %d %d\n ",
                m, n);
                        m*=2;n*=2;
                        printf("3 valeur de m et n dans le fils : %d %d\n ",
                m, n);
                        sleep(3);
                        exit(0);
    default:                                /*on est dans le processus père*/
                        sleep(2);
                        printf("4 valeur de m et n dans le père : %d %d\n ",
                m, n);
                        m*=3;n=*3;
                        printf("5 valeur de m et n dans le père : %d %d\n ",
                m, n);
                        sleep(2);
                        exit(0);}
                        }
```

- Suivez l'évolution des valeurs des variables m, n, qu'est ce que vous remarquer ?
- Comment les variables sont communiqués entre le père et le fils. Justifier.

Partie III : Les Pipes :

La présente partie a pour objectif la mise en place d'un moyen de communication entre les processus appartenant à la même famille. Pour cela nous proposons l'utilisation des pipes vus en cours.

Exécuter le programme suivant. Avant de répondre aux questions qui suivent**.**

```
#include <stdio.h>
#include <sys/signal.h>

main()
{int fils1, fils2, n, m, p[2]; char buf[5];
        pipe(p);                                /* création de pipe */
        if (fils1=fork()==0)                    /* création du premier
fils */
        {       printf("je suis le fils producteur \n");
                printf("j'écris 5 caractères dans le pipe \n");
                write(p[1],"ABCDE",5);
                printf("fin d'écriture dans le pipe \n");
                exit(3);}
        else                                    /* le père crée le fils
consommateur */
        {       if (fils2=fork()==0)            /* création du deuxième
```

```
            fils */
                        {       printf("je suis le fils consommateur \n");
                                read(p[0],buf,5);       /* lecture du pipe */
                                printf("voici les caractères lus \n");
                                write(1,buf,5);/*affichage des caractères sur
output standard*/
                                printf("\n");
                                exit(3);}
                        else{   printf("processus père c'est fini .... \n");
                                wait(&n);
                                wait(&m);}
}}
```

1. Combien de processus existent dans le code. Identifier pour chaque processus la partie d'exécution.
2. Remplir le pipe jusqu'à dépasser sa taille. Qu'est ce que vous remarquez ? Expliquer.
3. Modifier le code afin de lire à partir d'un pipe vide. Qu'est ce que vous remarquez ? Expliquer.
4. Modifier le code pour lire deux fois les mêmes données du pipe. Qu'est ce que vous remarquez ? Expliquer.
5. Modifier le programme pour que le père envoi un message au premier fils.
6. Ecrire un programme (prog4.c) qui permet :

   - La création de deux fils (fils 1 et fils 2).
   - La création d'un processus fils (fils 3) de fils1.
   - Le fils 3 crée un pipe avant d'envoyer un message au grand père (main). Qu'est ce que vous remarquez ? Expliquer.